**Contact form without side section**"use client"

import React, { useState } from "react"

import emailjs, { EmailJSResponseStatus } from "emailjs-com"

import TopNavOne from "@/components/Header/TopNav/TopNavOne"

import MenuOne from "@/components/Header/Menu/MenuOne"

import BreadcrumbItem from "@/components/Breadcrumb/BreadcrumbItem"

import CtaOne from "@/components/Section/CTA/CtaOne"

import Footer from "@/components/Footer/Footer"

import \* as Icon from "@phosphor-icons/react/dist/ssr"

export default function ContactStyleOne() {

  const [isHovered, setIsHovered] = useState(false)

  const [formData, setFormData] = useState({

    name: "",

    phone: "",

    email: "",

    selectedFile: null as File | null, // Explicitly type as File | null

    message: "",

    address: "",

  })

  const [successMessage, setSuccessMessage] = useState("") // State for success message

  const [errorMessage, setErrorMessage] = useState("") // State for error message

  const handleChange = (

    e: React.ChangeEvent<

      HTMLInputElement | HTMLTextAreaElement | HTMLSelectElement

    >

  ) => {

    setFormData({ ...formData, [e.target.name]: e.target.value })

  }

  // Handle file upload change (store the file object)

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {

    const file = e.target.files?.[0] || null

    setFormData({ ...formData, selectedFile: file })

  }

  // Convert file to Base64

  const getBase64 = (file: File) => {

    return new Promise<string>((resolve, reject) => {

      const reader = new FileReader()

      reader.onloadend = () => resolve(reader.result as string)

      reader.onerror = (error) => reject(error)

      reader.readAsDataURL(file)

    })

  }

  // Form submission handler

  const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {

    e.preventDefault()

    if (!formData.selectedFile) {

      setErrorMessage("Please select a file to upload.")

      return

    }

    // Convert the selected file to Base64

    const fileBase64 = await getBase64(formData.selectedFile)

    // EmailJS service details

    const serviceID = "your\_service\_id"

    const templateID = "your\_template\_id"

    const userID = "your\_user\_id"

    // Prepare the data for EmailJS

    const templateParams = {

      name: formData.name,

      phone: formData.phone,

      email: formData.email,

      message: formData.message,

      address: formData.address,

      file: fileBase64, // Base64 file content

      fileName: formData.selectedFile.name, // Send file name if necessary

    }

    // Send the email using EmailJS

    emailjs.send(serviceID, templateID, templateParams, userID).then(

      (response: EmailJSResponseStatus) => {

        // If email is sent successfully

        console.log("Email sent successfully", response)

        setSuccessMessage("Your details have been submitted successfully!")

        setFormData({

          name: "",

          phone: "",

          email: "",

          selectedFile: null, // Reset file

          message: "",

          address: "",

        })

        setTimeout(() => setSuccessMessage(""), 3000) // Clear success message after 3 seconds

      },

      (error: Error) => {

        // If there is an error sending the email

        console.error("Error sending email:", error)

        setErrorMessage("Failed to send your message. Please try again.")

      }

    )

  }

  return (

    <>

      <div className="overflow-x-hidden">

        <header id="header">

          <MenuOne />

        </header>

        <main className="content">

          <BreadcrumbItem

            link="Contact us"

            img="/images/banner/about.jpg"

            title="Contact us"

            desc="Explore our Contact Us page for prompt assistance from our dedicated team."

          />

          <div className="form-contact style-one lg:py-[100px] sm:py-16 py-10">

            <div className="container flex items-center justify-center">

              <div className="xm:w-5/6 w-full flex max-xl:flex-col xl:items-center gap-y-8">

                {/\* Left Side Section (Address, Phone, Email) Removed \*/}

                <div className="w-full xl:w-3/5 xl:pl-20">

                  <form

                    onSubmit={handleSubmit}

                    className="form-block flex flex-col justify-between gap-5"

                  >

                    <div className="heading">

                      <div className="heading5">Request a quote</div>

                      <div className="body3 text-secondary mt-2">

                        We will get back to you within 24 hours, or call us

                        every day

                      </div>

                    </div>

                    <div className="grid sm:grid-cols-2 gap-5">

                      <div className="w-full">

                        <input

                          name="name"

                          className="w-full bg-surface text-secondary caption1 px-4 py-3 rounded-lg"

                          type="text"

                          placeholder="Name"

                          value={formData.name}

                          onChange={handleChange}

                          required

                        />

                      </div>

                      <div className="w-full">

                        <input

                          name="phone"

                          className="w-full bg-surface text-secondary caption1 px-4 py-3 rounded-lg"

                          type="number"

                          placeholder="Phone No"

                          value={formData.phone}

                          onChange={handleChange}

                          required

                        />

                      </div>

                      <div className="col-span-2">

                        <input

                          name="email"

                          className="w-full bg-surface text-secondary caption1 px-4 py-3 rounded-lg"

                          type="email"

                          placeholder="Email"

                          value={formData.email}

                          onChange={handleChange}

                          required

                        />

                      </div>

                      <div className="col-span-2">

                        <input

                          type="file"

                          name="selectedFile"

                          className="w-full bg-surface text-secondary caption1 pl-3 py-3 rounded-lg"

                          onChange={handleFileChange}

                          required

                        />

                      </div>

                      {/\* New Address Field \*/}

                      <div className="col-span-2 w-full">

                        <input

                          name="address"

                          className="w-full bg-surface text-secondary caption1 px-4 py-3 rounded-lg"

                          type="text"

                          placeholder="Your Address"

                          value={formData.address}

                          onChange={handleChange}

                          required

                        />

                      </div>

                      <div className="col-span-2 w-full">

                        <textarea

                          name="message"

                          className="w-full bg-surface text-secondary caption1 px-4 py-3 rounded-lg"

                          rows={4}

                          placeholder="Type your Case"

                          value={formData.message}

                          onChange={handleChange}

                          required

                        ></textarea>

                      </div>

                    </div>

                    <div className="button-block">

                      <button

                        type="submit"

                        className="button-main text-white text-button rounded-full"

                        style={{

                          backgroundColor: isHovered ? "#0597fc" : "#050f3f", // Adjust the hover color as needed

                        }}

                        onMouseEnter={() => setIsHovered(true)}

                        onMouseLeave={() => setIsHovered(false)}

                      >

                        Submit request

                      </button>

                    </div>

                    {successMessage && (

                      <div style={{ color: "green" }} className="mt-2">

                        {successMessage}

                      </div>

                    )}

                    {errorMessage && (

                      <div style={{ color: "red" }} className="mt-2">

                        {errorMessage}

                      </div>

                    )}

                  </form>

                </div>

              </div>

            </div>

          </div>

        </main>

        <footer id="footer">

          <Footer />

        </footer>

      </div>

    </>

  )

}

/\*

To modify the existing component so that when a user fills out the contact form and submits it, their details are sent directly to your email (instead of storing them in Firebase), you can achieve this by using a third-party email service, such as \*\*EmailJS\*\* or \*\*SMTP\*\*.

Since you're working in a Next.js app and would prefer not to make changes to the UI (as you mentioned, keeping it the same), here's how you can set this up using \*\*EmailJS\*\* for simplicity:

### Steps:

1. \*\*Sign up for EmailJS\*\*:

   - Go to [EmailJS](https://www.emailjs.com/).

   - Create an account.

   - Follow the instructions to create an email service (like Gmail, Outlook, etc.).

   - Create an email template for the contact form. The template will define how the email looks and the data you want to receive.

2. \*\*Install EmailJS SDK\*\*:

   In your Next.js project, you'll need to install the EmailJS SDK:

   ```bash

   npm install emailjs-com

   ```

3. \*\*Configure EmailJS in your Component\*\*:

   After setting up the service and template on EmailJS, use their SDK to send the data from the contact form to your email.

Here is the updated code for the contact form, which will send form data directly to your email using \*\*EmailJS\*\*.

### Updated Component Code:

```tsx

"use client"

import React, { useState } from "react"

import emailjs from "emailjs-com"

import TopNavOne from "@/components/Header/TopNav/TopNavOne"

import MenuOne from "@/components/Header/Menu/MenuOne"

import BreadcrumbItem from "@/components/Breadcrumb/BreadcrumbItem"

import CtaOne from "@/components/Section/CTA/CtaOne"

import Footer from "@/components/Footer/Footer"

import \* as Icon from "@phosphor-icons/react/dist/ssr"

export default function ContactStyleOne() {

  const [isHovered, setIsHovered] = useState(false)

  const [formData, setFormData] = useState({

    name: "",

    phone: "",

    email: "",

    loanType: "Select your Loan", // Default to 'Select your Loan'

    message: "",

    address: "", // New field for address

  })

  const [successMessage, setSuccessMessage] = useState("") // State for success message

  const [errorMessage, setErrorMessage] = useState("") // State for error message

  const handleChange = (

    e: React.ChangeEvent<

      HTMLInputElement | HTMLTextAreaElement | HTMLSelectElement

    >

  ) => {

    setFormData({ ...formData, [e.target.name]: e.target.value })

  }

  // Form submission handler

  const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {

    e.preventDefault()

    // EmailJS service details

    const serviceID = "your\_service\_id"

    const templateID = "your\_template\_id"

    const userID = "your\_user\_id"

    // Prepare the data for EmailJS

    const templateParams = {

      name: formData.name,

      phone: formData.phone,

      email: formData.email,

      loanType: formData.loanType,

      message: formData.message,

      address: formData.address,

    }

    // Send the email

    emailjs

      .send(serviceID, templateID, templateParams, userID)

      .then(

        () => {

          // If successful

          setSuccessMessage("Your details have been submitted successfully!")

          setFormData({

            name: "",

            phone: "",

            email: "",

            loanType: "Select your Loan", // Reset loan type

            message: "",

            address: "", // Reset address field

          })

          setTimeout(() => setSuccessMessage(""), 3000) // Clear message after 3 seconds

        },

        (error) => {

          // If failed

          console.error("Error sending email:", error)

          setErrorMessage("Failed to send your message. Please try again.")

        }

      )

  }

  return (

    <>

      <div className="overflow-x-hidden">

        <header id="header">

          <TopNavOne />

          <MenuOne />

        </header>

        <main className="content">

          <BreadcrumbItem

            link="Contact us"

            img="/images/banner/about.jpg"

            title="Contact us"

            desc="Explore our Contact Us page for prompt assistance from our dedicated team."

          />

          <div className="form-contact style-one lg:py-[100px] sm:py-16 py-10">

            <div className="container flex items-center justify-center">

              <div className="xm:w-5/6 w-full flex max-xl:flex-col xl:items-center gap-y-8">

### Explanation:

1. \*\*EmailJS Setup\*\*:

   - Replace `your\_service\_id`, `your\_template\_id`, and `your\_user\_id` with the values from your EmailJS account.

   - The `templateParams` object holds the form data, and it's sent to EmailJS to generate an email based on the template you created.

2. \*\*Form Submission\*\*:

   - When the form is submitted, the data is sent via the `emailjs.send

\*/

**Code for install translation**  
Great! Since your site is static and you don't have dynamic content coming from a database or API, \*\*Static Site Generation (SSG)\*\* is the most efficient approach for implementing multi-language support in a Next.js project.

Here’s a step-by-step guide for setting up \*\*language support\*\* in a \*\*static\*\* Next.js site using \*\*`next-i18next`\*\*:

---

### 1. \*\*Install Dependencies\*\*

You’ll need `next-i18next` to handle internationalization. Install the necessary packages:

```bash

npm install next-i18next react-i18next

```

---

### 2. \*\*Configure `next-i18next`\*\*

Now, set up `next-i18next` by creating a `next-i18next.config.js` file to configure the supported languages.

#### Create `next-i18next.config.js`:

```js

// next-i18next.config.js

module.exports = {

i18n: {

defaultLocale: "en", // Default language (English)

locales: ["en", "ta"], // Available languages (English and Tamil)

},

reloadOnPrerender: process.env.NODE\_ENV === "development", // Enable reloading during development

}

```

---

### 3. \*\*Create Translation Files\*\*

Create the translation files for each language in the `locales` folder. You can structure the translations in JSON files.

#### Create `locales/en/common.json` (English translations):

```json

{

"about\_title": "About Us",

"section1\_content": "Founded in 2009 by Mr. Jaya Prakash..."

}

```

#### Create `locales/ta/common.json` (Tamil translations):

```json

{

"about\_title": "எங்களை பற்றி",

"section1\_content": "2009 இல் திரு. ஜெயா பிரகாஷ் என்பவரால் நிறுவப்பட்டது..."

}

```

You can add more translation keys as needed.

---

### 4. \*\*Modify `next.config.js` to include `next-i18next`\*\*

Next, modify the `next.config.js` file to include the `next-i18next` configuration.

```js

// next.config.js

const nextTranslate = require("next-translate")

module.exports = nextTranslate({

reactStrictMode: true,

})

```

This makes sure that the `next-i18next` plugin is applied.

---

### 5. \*\*Implement the Language Toggle Button\*\*

You can create a separate `LanguageToggle` component that lets users switch between languages.

#### Create `components/LanguageToggle.tsx`:

```tsx

// components/LanguageToggle.tsx

"use client"

import React from "react"

import { useTranslation } from "react-i18next"

const LanguageToggle = () => {

const { i18n } = useTranslation()

const toggleLanguage = () => {

const nextLang = i18n.language === "en" ? "ta" : "en"

i18n.changeLanguage(nextLang)

}

return (

<button

onClick={toggleLanguage}

style={{

backgroundColor: "#007BFF",

color: "#fff",

padding: "10px 20px",

borderRadius: "5px",

}}

>

{i18n.language === "en" ? "Switch to Tamil" : "Switch to English"}

</button>

)

}

export default LanguageToggle

```

This component will toggle the language between English and Tamil. The text will change dynamically based on the current language.

---

### 6. \*\*Use Translations in Your Pages\*\*

Now, you can use the `useTranslation` hook in your pages to display translations.

#### Modify `pages/index.tsx` to use translations:

```tsx

// pages/index.tsx

import { useTranslation } from "react-i18next"

import LanguageToggle from "../components/LanguageToggle"

const Home = () => {

const { t } = useTranslation()

return (

<div>

<h1>{t("common:about\_title")}</h1>

<p>{t("common:section1\_content")}</p>

<div style={{ marginTop: "20px" }}>

<LanguageToggle />

</div>

</div>

)

}

export default Home

```

The `t("common:about\_title")` and `t("common:section1\_content")` are placeholders for the translations from the respective language JSON files (`common.json` for both languages).

---

### 7. \*\*Static Site Generation with Translations\*\*

In static sites, you need to use \*\*`getStaticProps`\*\* to preload the translation files at build time.

#### Modify `pages/index.tsx` to include `getStaticProps`:

```tsx

// pages/index.tsx

import { useTranslation } from "react-i18next"

import { GetStaticProps } from "next"

import { serverSideTranslations } from "next-i18next/serverSideTranslations"

import LanguageToggle from "../components/LanguageToggle"

const Home = () => {

const { t } = useTranslation()

return (

<div>

<h1>{t("common:about\_title")}</h1>

<p>{t("common:section1\_content")}</p>

<div style={{ marginTop: "20px" }}>

<LanguageToggle />

</div>

</div>

)

}

// Static generation of pages with translations

export const getStaticProps: GetStaticProps = async ({ locale }) => {

return {

props: {

...(await serverSideTranslations(locale, ["common"])),

},

}

}

export default Home

```

- \*\*`getStaticProps`\*\*: Fetches the necessary translations during build time.

- \*\*`serverSideTranslations`\*\*: This helper function from `next-i18next` preloads the translations for the current locale (`en` or `ta`).

This will ensure that the page is generated with the correct translations.

---

### 8. \*\*Build and Run the Application\*\*

Now, you can run the build process to generate the static pages with the translations.

1. \*\*Build the project\*\*:

```bash

npm run build

```

2. \*\*Start the application\*\*:

```bash

npm start

```

3. \*\*Test your Language Toggle\*\*:

- Visit your site and you should see the content in \*\*English\*\* by default.

- When you click the "Switch to Tamil" button, the content should change to Tamil, and vice versa.

---

### 9. \*\*Optional: Deploying to Vercel or Static Hosting\*\*

Once your site is ready and fully translated, you can deploy it to platforms like \*\*Vercel\*\* or any static hosting provider that supports Next.js, such as \*\*Netlify\*\*.

- Deploy to \*\*Vercel\*\*: If you use Vercel, deployment is very straightforward, as it integrates directly with Next.js.

- Other static hosts: Make sure the build output is correct for your chosen static hosting provider.

---

### Summary

Since your website is static, the \*\*best approach\*\* is to use \*\*Static Site Generation (SSG)\*\* along with `next-i18next`. This method ensures that translations are preloaded at build time and are SEO-friendly, since they are available in the HTML when the page is loaded.

- \*\*`next-i18next`\*\* handles the language switching and ensures the correct translations are available.

- \*\*Static generation with `getStaticProps`\*\* ensures that your pages are built with the correct translations for each language at build time.

- \*\*Language toggle button\*\* can be used to switch between English and Tamil.

With this approach, your site will be fully static, with translations baked into the build, offering fast load times and a smooth user experience.